Really fast syndrome-based hashing

Peter Schwabe

National Taiwan University

Joint work with Daniel J. Bernstein, Tanja Lange, Christiane Peters

July 5, 2011

Africacrypt 2011, Dakar, Senegal

Introduction – Hash functions

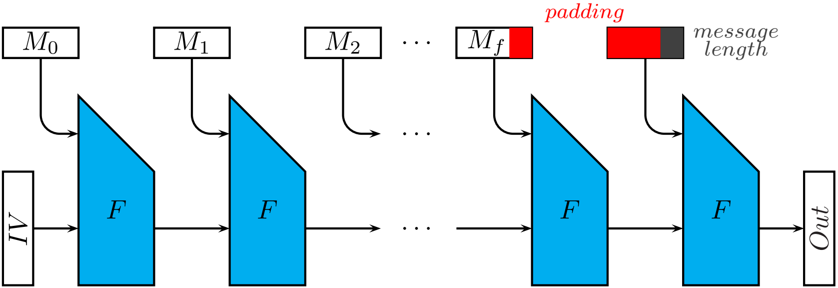
- Function $h : \{0,1\}^* \to \{0,1\}^n$
- Preimage resistance: Given $h(M)$, infeasible to find M
- Second preimage resistance: Given M, infeasible to find $M' \neq M$ with $h(M) = h(M')$
- Collision resistance: Infeasible to find M, M', with $M \neq M'$ and $h(M) = h(M')$

Introduction – Hash functions

- Function $h : \{0,1\}^* \to \{0,1\}^n$
- Preimage resistance: Given $h(M)$, infeasible to find M
- Second preimage resistance: Given M, infeasible to find $M' \neq M$ with $h(M) = h(M')$
- Collision resistance: Infeasible to find M, M', with $M \neq M'$ and $h(M) = h(M')$
- "Trivial" property: Hash functions irreversibly compress arbitrarily long strings
- Arbitrarily long usully means: Some sort of iterative process

Merkle-Damgård iteration

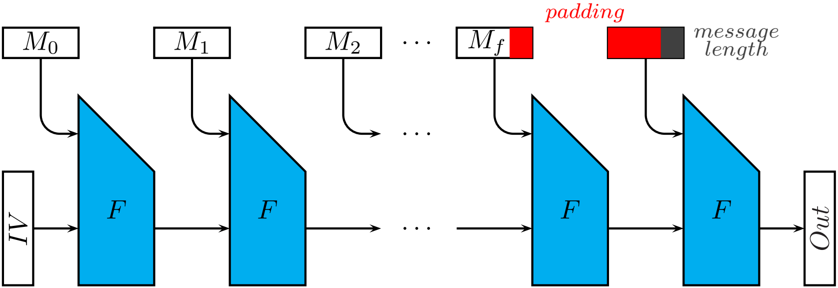
- Use fixed-input-length compression function $F : \{0,1\}^\ell \to \{0,1\}^k$ with $\ell > k$



- Apply output filter $\{0,1\}^k \to \{0,1\}^n$

Merkle-Damgård iteration

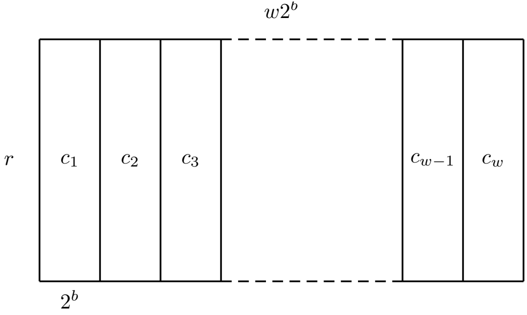
- Use fixed-input-length compression function $F : \{0,1\}^\ell \to \{0,1\}^k$ with $\ell > k$



- Apply output filter $\{0,1\}^k \to \{0,1\}^n$
- In the following: Zoom into F

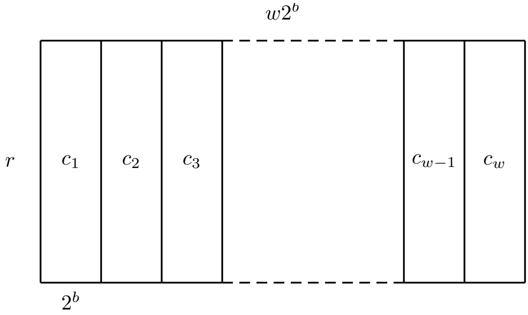
A simple construction for F

- Consider input of length $\ell = w \cdot b$, hence, $m = (m_1, m_2, \dots, m_w)$, each m_i with b bits
- Take an $r \times w2^b$ binary (pseudo-)random matrix, decomposed into w blocks with 2^b columns each: $C = (c_1, c_2, \dots, c_w)$



- Define $F(m) = c_1[m_1] \oplus c_2[m_2] \oplus \cdots \oplus c_w[m_w]$

How about collisions?



- Resistance obviously depends on b, w, and r
- Larger r makes it harder to find collisions (but reduces compression factor)
- Smaller w or b makes it harder to find collisions (but reduces compression factor)

Specifying the parameters

- Long history of compression functions with similar constructions
- . . . also long history of breaks (see paper)
- In the remainder of this talk consider:

Specifying the parameters

- Long history of compression functions with similar constructions
- ... also long history of breaks (see paper)
- In the remainder of this talk consider:

FSB-256

- FSB is a SHA-3 round-1 candidate by Augot, Finiasz, Gaborit, Manuel, Sendrier
- FSB-256 is designed to provide 2^{128} bits of security against collisions
- Parameters: $b = 14$, $w = 128$, $r = 1024$

Specifying the parameters

- Long history of compression functions with similar constructions
- ... also long history of breaks (see paper)
- In the remainder of this talk consider:

FSB-256

- FSB is a SHA-3 round-1 candidate by Augot, Finiasz, Gaborit, Manuel, Sendrier
- FSB-256 is designed to provide 2^{128} bits of security against collisions
- Parameters: $b = 14$, $w = 128$, $r = 1024$

RFSB-509

- RFSB is the family of compression functions described in this paper
- RFSB-509 is designed to provide 2^{128} bits of security against collisions
- Parameters: $b = 8$, $w = 112$, $r = 509$

FSB-256 performance

- FSB is unbroken, but did not make it to round-2 of the SHA-3 competition
- Reason: It is too slow, 95.53 cycles/byte on an Intel Core 2 Quad Q9550
- Comparison: SHA-256 takes just 15.26 cycles/byte on the same machine

FSB-256 performance

- FSB is unbroken, but did not make it to round-2 of the SHA-3 competition
- Reason: It is too slow, 95.53 cycles/byte on an Intel Core 2 Quad Q9550
- Comparison: SHA-256 takes just 15.26 cycles/byte on the same machine
- Main problem: xors are fast, but loading matrix columns is slow for a large matrix

FSB-256 performance

- FSB is unbroken, but did not make it to round-2 of the SHA-3 competition
- Reason: It is too slow, 95.53 cycles/byte on an Intel Core 2 Quad Q9550
- Comparison: SHA-256 takes just 15.26 cycles/byte on the same machine
- Main problem: xors are fast, but loading matrix columns is slow for a large matrix
- Solution (in both FSB and RFSB): Use compressed matrix

FSB-256 performance

- FSB is unbroken, but did not make it to round-2 of the SHA-3 competition
- Reason: It is too slow, 95.53 cycles/byte on an Intel Core 2 Quad Q9550
- Comparison: SHA-256 takes just 15.26 cycles/byte on the same machine
- Main problem: xors are fast, but loading matrix columns is slow for a large matrix
- Solution (in both FSB and RFSB): Use compressed matrix
- Idea: Consider columns as elements of $\mathbb{F}_2[X]/(X^p - 1)$

FSB-256 performance

- FSB is unbroken, but did not make it to round-2 of the SHA-3 competition
- Reason: It is too slow, 95.53 cycles/byte on an Intel Core 2 Quad Q9550
- Comparison: SHA-256 takes just 15.26 cycles/byte on the same machine
- Main problem: xors are fast, but loading matrix columns is slow for a large matrix
- Solution (in both FSB and RFSB): Use compressed matrix
- Idea: Consider columns as elements of $\mathbb{F}_2[X]/(X^p - 1)$
- For FSB use $p = 1061$ and construct $c_i[0], c_i[1], \dots, c_i[16383]$ as

$$
\begin{array}{cccc}
c_i[0], & c_i[0]X, & c_i[0]X^2, \dots, & c_i[0]X^{1023}, \\
c_i[1024], & c_i[1024]X, & c_i[1024]X^2, \dots, & c_i[1024]X^{1023}, \\
\dots \\
\multicolumn{4}{c}{c_i[15360], c_i[15360]X, c_i[15360]X^2, \dots, c_i[15360]X^{1023}}
\end{array}
$$

FSB-256 performance

- FSB is unbroken, but did not make it to round-2 of the SHA-3 competition
- Reason: It is too slow, 95.53 cycles/byte on an Intel Core 2 Quad Q9550
- Comparison: SHA-256 takes just 15.26 cycles/byte on the same machine
- Main problem: xors are fast, but loading matrix columns is slow for a large matrix
- Solution (in both FSB and RFSB): Use compressed matrix
- Idea: Consider columns as elements of $\mathbb{F}_2[X]/(X^p - 1)$
- For FSB use $p = 1061$ and construct $c_i[0], c_i[1], \ldots, c_i[16383]$ as

$$\begin{array}{cccc} c_i[0], & c_i[0]X, & c_i[0]X^2, \ldots, & c_i[0]X^{1023}, \\ c_i[1024], & c_i[1024]X, & c_i[1024]X^2, \ldots, & c_i[1024]X^{1023}, \\ \ldots \\ \multicolumn{4}{c}{c_i[15360], c_i[15360]X, c_i[15360]X^2, \ldots, c_i[15360]X^{1023}} \end{array}$$

- Note that rotation distances (exponents of X) depend on input

RFSB-509 performance

- Benchmarks from RFSB-509 with SHA-256 output filter
- 10.67 cycles/byte on the same Intel Core 2 Quad Q9550

RFSB-509 performance

- Benchmarks from RFSB-509 with SHA-256 output filter
- 10.67 cycles/byte on the same Intel Core 2 Quad Q9550

Why is RFSB-509 so much faster?

- Compression of the table through $c_i[j]$ as $c[j]X^i$ (or rather $c[j]X^{128(w-i)}$) instead of $c_i[0]X^j$: **fixed rotation distances**

RFSB-509 performance

- Benchmarks from RFSB-509 with SHA-256 output filter
- 10.67 cycles/byte on the same Intel Core 2 Quad Q9550

Why is RFSB-509 so much faster?

- Compression of the table through $c_i[j]$ as $c[j]X^i$ (or rather $c[j]X^{128(w-i)}$) instead of $c_i[0]X^j$: **fixed rotation distances**
- Smaller compressed table, 16,384 bytes instead of at least 262,144 bytes (fast implementations of FSB use larger precomputed matrix parts): matrix **fits into L1 cache** of most modern microprocessors

RFSB-509 performance

- Benchmarks from RFSB-509 with SHA-256 output filter
- 10.67 cycles/byte on the same Intel Core 2 Quad Q9550

Why is RFSB-509 so much faster?

- Compression of the table through $c_i[j]$ as $c[j]X^i$ (or rather $c[j]X^{128(w-i)}$) instead of $c_i[0]X^j$: **fixed rotation distances**
- Smaller compressed table, 16,384 bytes instead of at least 262,144 bytes (fast implementations of FSB use larger precomputed matrix parts): matrix **fits into L1 cache** of most modern microprocessors
- Hand-optimized assembly implementation (for AMD64)
- Implementation-aware design

Why is our matrix smaller?

- Short answer: tighter security analysis

Why is our matrix smaller?

- Short answer: tighter security analysis
- Three types of attacks against FSB/RFSB:
  1. Linearization attacks
  2. Generalized birthday attacks
  3. Information-set decoding

Why is our matrix smaller?

- Short answer: tighter security analysis
- Three types of attacks against FSB/RFSB:
  1. Linearization attacks
  2. Generalized birthday attacks
  3. Information-set decoding
- FSB designers overestimated the power of 2. and 3.

Why is our matrix smaller?

- Short answer: tighter security analysis
- Three types of attacks against FSB/RFSB:
  1. Linearization attacks
  2. Generalized birthday attacks
  3. Information-set decoding
- FSB designers overestimated the power of 2. and 3.
- Let's look at ways to generate collisions, i.e. $2w$ columns, 2 per block, that add up to zero

The power of birthday attacks

- Idea: Start with 2^t lists containing (sums of) columns, proceed in various levels:
- In each level obtain 2^{i-1} lists from 2^i lists through merging
- List length remains constant, each merging eliminates bits of the entries
- In the last step, find two vectors adding up to zero

The power of birthday attacks

- Idea: Start with 2^t lists containing (sums of) columns, proceed in various levels:
- In each level obtain 2^{i-1} lists from 2^i lists through merging
- List length remains constant, each merging eliminates bits of the entries
- In the last step, find two vectors adding up to zero
- Problem: huge storage requirements
- Many estimates ignore the machine size, assume $O(1)$ for access to memory of arbitrary size

The power of birthday attacks

- Idea: Start with 2^t lists containing (sums of) columns, proceed in various levels:
- In each level obtain 2^{i-1} lists from 2^i lists through merging
- List length remains constant, each merging eliminates bits of the entries
- In the last step, find two vectors adding up to zero
- Problem: huge storage requirements
- Many estimates ignore the machine size, assume $O(1)$ for access to memory of arbitrary size
- Indocrypt 2009 paper by Bernstein, Lange, Niederhagen, Peters, and Schwabe presented highly optimized generalized birthday attack against FSB-48 compression function (toy version of FSB)
- This attack took 7 days, 23 hours and 53 minutes on 8 quad-core machines, using > 5 TB of storage
- Comparison: breaking the FSB-48 *hash function* takes less than 2 minutes on one core of one of the machines with negligible storage

The power of birthday attacks

- Idea: Start with 2^t lists containing (sums of) columns, proceed in various levels:
- In each level obtain 2^{i-1} lists from 2^i lists through merging
- List length remains constant, each merging eliminates bits of the entries
- In the last step, find two vectors adding up to zero
- Problem: huge storage requirements
- Many estimates ignore the machine size, assume $O(1)$ for access to memory of arbitrary size
- Indocrypt 2009 paper by Bernstein, Lange, Niederhagen, Peters, and Schwabe presented highly optimized generalized birthday attack against FSB-48 compression function (toy version of FSB)
- This attack took 7 days, 23 hours and 53 minutes on 8 quad-core machines, using > 5 TB of storage
- Comparison: breaking the FSB-48 *hash function* takes less than 2 minutes on one core of one of the machines with negligible storage
- Compression functions of full FSB versions are similarly over-dimensioned

The power of information-set decoding

- Algorithm from coding theory, find low-weight code words
- In the context of FSB/RFSB: Find $2w$ columns adding up to zero
- Problem: Very low probability to have exactly two columns per block

The power of information-set decoding

- Algorithm from coding theory, find low-weight code words
- In the context of FSB/RFSB: Find $2w$ columns adding up to zero
- Problem: Very low probability to have exactly two columns per block
- FSB is designed to protect against ISD

The power of information-set decoding

- Algorithm from coding theory, find low-weight code words
- In the context of FSB/RFSB: Find $2w$ columns adding up to zero
- Problem: Very low probability to have exactly two columns per block
- FSB is designed to protect against ISD
- RFSB is designed to protect against 2-regular ISD
- We presented improved 2-regular ISD at IWCC 2011
- RFSB is protected against this new improved algorithm

The power of information-set decoding

- Algorithm from coding theory, find low-weight code words
- In the context of FSB/RFSB: Find $2w$ columns adding up to zero
- Problem: Very low probability to have exactly two columns per block
- FSB is designed to protect against ISD
- RFSB is designed to protect against 2-regular ISD
- We presented improved 2-regular ISD at IWCC 2011
- RFSB is protected against this new improved algorithm
- Finding 2-regular low-weight codewords is not as well studied as finding general low-weight codewords
- We encourage the community to try to improve our complexity bounds

More in the paper

- Full specification of RFSB and RFSB-509 (including matrix generation)
- Some more history of designs and breaks
- Detailed description of the implementation
- Extra speed: incremental hashing
- Extra speed: fast batch verification of hashes
- Extra security: Elimination of variable-index table lookups (at the expense of speed)
- Detailed attack analysis with some new generalizations

Conclusion

- RFSB-509 is faster than 7 out of 14 SHA-3 round-2 candidates
- RFSB-509 is faster than 3 out of 5 SHA-3 finalists
- Software is in the public domain, submitted to eBASH for public benchmarking

Paper online: http://eprint.iacr.org/2011/074/